Subject: [PATCH 0/14 (3 subsets)] Make tuns and vlans devices work per-net.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:36:49 GMT
View Forum Message <> Reply to Message

Hi, guys.

I've recently sent a TUN devices virtualization, but it was rejected
by Dave, since the struct net is becoming a dumping ground.

I agree with him - we really need some way to register on-net data
dynamically. That's my view of such a thing and two examples of how
to use it (TUN and VLAN devices virtualization).

If this will be found good, I'll send these sets to David, hoping he
will accept them :)

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 1/14][NETNS]: Introduce the net-subsys id generator.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:40:44 GMT
View Forum Message <> Reply to Message

To make some per-net generic pointers, we need some way to
address them, i.e. - IDs. This is simple IDA-based IDs
generator for pernet subsystems. They will be used in the
next patches.

Since it will be used by devices only (tun and vlan), I make
it resemble the register_pernet_device functionality.

The new ids is stored in the *id pointer _before_ calling the
init callback to make this id available in this callback.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 include/net/net_namespace.h | 2 ++
 net/core/net_namespace.c | 36 ++++++++++++++++++++++++++++++++++++
 2 files changed, 38 insertions(+), 0 deletions(-)

diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
index 0ab62ed..6971fdb 100644
--- a/include/net/net_namespace.h

```
+++ b/include/net/net_namespace.h
@@ -181,6 +181,8 @@ extern int register_pernet_subsys(struct pernet_operations *);
 extern void unregister_pernet_subsys(struct pernet_operations *);
 extern int register_pernet_device(struct pernet_operations *);
 extern void unregister_pernet_device(struct pernet_operations *);
+extern int register_pernet_gen_device(int *id, struct pernet_operations *);
+extern void unregister_pernet_gen_device(int id, struct pernet_operations *);

 struct ctl_path;
 struct ctl_table;
diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
index 7b66083..7ef3bac 100644
--- a/net/core/net_namespace.c
+++ b/net/core/net_namespace.c
@@ -5,6 +5,7 @@
 #include <linux/list.h>
 #include <linux/delay.h>
 #include <linux/sched.h>
+#include <linux/idr.h>
 #include <net/net_namespace.h>

 /*
@@ -253,6 +254,8 @@ static void unregister_pernet_operations(struct pernet_operations *ops)
 }
 #endif

+static DEFINE_IDA(net_generic_ids);
+
 /**
  *      register_pernet_subsys - register a network namespace subsystem
  * @ops:  pernet operations structure for the subsystem
@@ -330,6 +333,28 @@ int register_pernet_device(struct pernet_operations *ops)
 }
 EXPORT_SYMBOL_GPL(register_pernet_device);

+int register_pernet_gen_device(int *id, struct pernet_operations *ops)
+{
+ int error;
+ mutex_lock(&net_mutex);
+again:
+ error = ida_get_new_above(&net_generic_ids, 1, id);
+ if (error) {
+  if (error == -EAGAIN) {
+   ida_pre_get(&net_generic_ids, GFP_KERNEL);
+   goto again;
+  }
+ }
+ error = register_pernet_operations(first_device, ops);
```

```
+ if (error)
+  ida_remove(&net_generic_ids, *id);
+ else if (first_device == &pernet_list)
+  first_device = &ops->list;
+ mutex_unlock(&net_mutex);
+ return error;
+}
+EXPORT_SYMBOL_GPL(register_pernet_gen_device);
+
 /**
  *      unregister_pernet_device - unregister a network namespace netdevice
  * @ops: pernet operations structure to manipulate
@@ -348,3 +373,14 @@ void unregister_pernet_device(struct pernet_operations *ops)
  mutex_unlock(&net_mutex);
 }
 EXPORT_SYMBOL_GPL(unregister_pernet_device);
+
+void unregister_pernet_gen_device(int id, struct pernet_operations *ops)
+{
+ mutex_lock(&net_mutex);
+ if (&ops->list == first_device)
+  first_device = first_device->next;
+ unregister_pernet_operations(ops);
+ ida_remove(&net_generic_ids, id);
+ mutex_unlock(&net_mutex);
+}
+EXPORT_SYMBOL_GPL(unregister_pernet_gen_device);
--
1.5.3.4
```

Subject: [PATCH 2/14][NETNS]: Generic per-net pointers.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:43:30 GMT
View Forum Message <> Reply to Message

Add the elastic array of void * pointer to the struct net.
The access rules are simple:

1. register the ops with register_pernet_gen_device to get
   the id of your private pointer
2. call net_assign_generic() to put the private data on the
   struct net (most preferably this should be done in the
   ->init callback of the ops registered)

3. do not change this pointer while the net is alive;
4. use the net_generic() to get the pointer.

When adding a new pointer, I copy the old array, replace it
with a new one and schedule the old for kfree after an RCU
grace period.

Since the net_generic explores the net->gen array inside rcu
read section and once set the net->gen->ptr[x] pointer never
changes, this grants us a safe access to generic pointers.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
include/net/net_namespace.h | 2 +
include/net/netns/generic.h | 49 ++
net/core/net_namespace.c | 62 ++
3 files changed, 113 insertions(+), 0 deletions(-)
create mode 100644 include/net/netns/generic.h

diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
index 6971fdb..e3d4eb4 100644
--- a/include/net/net_namespace.h
+++ b/include/net/net_namespace.h
@@ -19,6 +19,7 @@ struct proc_dir_entry;
struct net_device;
struct sock;
struct ctl_table_header;
+struct net_generic;

struct net {
atomic_t count; /* To decided when the network
@@ -57,6 +58,7 @@ struct net {
#ifdef CONFIG_NETFILTER
struct netns_xt xt;
#endif
+ struct net_generic *gen;
};

diff --git a/include/net/netns/generic.h b/include/net/netns/generic.h
new file mode 100644
index 0000000..e8a6d27
--- /dev/null
+++ b/include/net/netns/generic.h
@@ -0,0 +1,49 @@
+/*
+ * generic net pointers

```
+ */
+
+#ifndef __NET_GENERIC_H__
+#define __NET_GENERIC_H__
+
+#include <linux/rcupdate.h>
+
+/*
+ * Generic net pointers are to be used by modules
+ * to put some private stuff on the struct net without
+ * explicit struct net modification
+ *
+ * The rules are simple:
+ * 1. register the ops with register_pernet_gen_device to get
+ *    the id of your private pointer
+ * 2. call net_assign_generic() to put the private data on the
+ *    struct net (most preferably this should be done in the
+ *    ->init callback of the ops registered)
+ * 3. do not change this pointer while the net is alive.
+ *
+ * After accomplishing all of the above, the private pointer
+ * can be accessed with the net_generic() call.
+ */
+
+struct net_generic {
+ unsigned int len;
+ struct rcu_head rcu;
+
+ void *ptr[0];
+};
+
+static inline void *net_generic(struct net *net, int id)
+{
+ struct net_generic *ng;
+ void *ptr;
+
+ rcu_read_lock();
+ ng = rcu_dereference(net->gen);
+ BUG_ON(id == 0 || id > ng->len);
+ ptr = ng->ptr[id - 1];
+ rcu_read_unlock();
+
+ return ptr;
+}
+
+extern int net_assign_generic(struct net *net, int id, void *data);
+#endif
diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
```

```
index 7ef3bac..b384840 100644
--- a/net/core/net_namespace.c
+++ b/net/core/net_namespace.c
@@ -7,6 +7,7 @@
 #include <linux/sched.h>
 #include <linux/idr.h>
 #include <net/net_namespace.h>
+#include <net/netns/generic.h>

 /*
  * Our network namespace constructor/destructor lists
@@ -21,6 +22,8 @@ LIST_HEAD(net_namespace_list);
 struct net init_net;
 EXPORT_SYMBOL(init_net);

+#define INITIAL_NET_GEN_PTRS 13 /* +1 for len +2 for rcu_head */
+
 /*
  * setup_net runs the initializers for the network namespace object.
  */
@@ -29,10 +32,21 @@ static __net_init int setup_net(struct net *net)
  /* Must be called with net_mutex held */
  struct pernet_operations *ops;
  int error;
+ struct net_generic *ng;

  atomic_set(&net->count, 1);
  atomic_set(&net->use_count, 0);

+ error = -ENOMEM;
+ ng = kzalloc(sizeof(struct net_generic) +
+   INITIAL_NET_GEN_PTRS * sizeof(void *), GFP_KERNEL);
+ if (ng == NULL)
+  goto out;
+
+ ng->len = INITIAL_NET_GEN_PTRS;
+ INIT_RCU_HEAD(&ng->rcu);
+ rcu_assign_pointer(net->gen, ng);
+
  error = 0;
  list_for_each_entry(ops, &pernet_list, list) {
   if (ops->init) {
@@ -54,6 +68,7 @@ out_undo:
  }

  rcu_barrier();
+ kfree(ng);
  goto out;
```

```
 }

@@ -384,3 +399,50 @@ void unregister_pernet_gen_device(int id, struct pernet_operations
*ops)
  mutex_unlock(&net_mutex);
 }
 EXPORT_SYMBOL_GPL(unregister_pernet_gen_device);
+
+static void net_generic_release(struct rcu_head *rcu)
+{
+ struct net_generic *ng;
+
+ ng = container_of(rcu, struct net_generic, rcu);
+ kfree(ng);
+}
+
+int net_assign_generic(struct net *net, int id, void *data)
+{
+ struct net_generic *ng, *old_ng;
+
+ BUG_ON(!mutex_is_locked(&net_mutex));
+ BUG_ON(id == 0);
+
+ ng = old_ng = net->gen;
+ if (old_ng->len >= id)
+  goto assign;
+
+ ng = kzalloc(sizeof(struct net_generic) +
+   id * sizeof(void *), GFP_KERNEL);
+ if (ng == NULL)
+  return -ENOMEM;
+
+ /*
+  * Some synchronisation notes:
+  *
+  * The net_generic explores the net->gen array inside rcu
+  * read section. Besides once set the net->gen->ptr[x]
+  * pointer never changes (see rules in netns/generic.h).
+  *
+  * That said, we simply duplicate this array and schedule
+  * the old copy for kfree after a grace period.
+  */
+
+ ng->len = id;
+ INIT_RCU_HEAD(&ng->rcu);
+ memcpy(&ng->ptr, &old_ng->ptr, old_ng->len);
+
+ rcu_assign_pointer(net->gen, ng);
```

```
+ call_rcu(&old_ng->rcu, net_generic_release);
+assign:
+ ng->ptr[id - 1] = data;
+ return 0;
+}
+EXPORT_SYMBOL_GPL(net_assign_generic);
--
1.5.3.4
```

Subject: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:46:15 GMT
View Forum Message <> Reply to Message

This is the first step in making tuntap devices work in net
namespaces. The structure mentioned is pointed by generic
net pointer with tun_net_id id, and tun driver fills one on
its load. It will contain only the tun devices list.

So declare this structure and introduce net init and exit hooks.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 drivers/net/tun.c |   53 ++++++++++++++++++++++++++++++++++++++++++++++++++++-
 1 files changed, 52 insertions(+), 1 deletions(-)

diff --git a/drivers/net/tun.c b/drivers/net/tun.c
index 7b816a0..9bfba02 100644
--- a/drivers/net/tun.c
+++ b/drivers/net/tun.c
@@ -63,6 +63,7 @@
 #include <linux/if_tun.h>
 #include <linux/crc32.h>
 #include <net/net_namespace.h>
+#include <net/netns/generic.h>

 #include <asm/system.h>
 #include <asm/uaccess.h>
@@ -73,6 +74,11 @@ static int debug;

 /* Network device part of the driver */
```

```
+static unsigned int tun_net_id;
+struct tun_net {
+ struct list_head dev_list;
+};
+
 static LIST_HEAD(tun_dev_list);
 static const struct ethtool_ops tun_ethtool_ops;

@@ -873,6 +879,37 @@ static const struct ethtool_ops tun_ethtool_ops = {
  .set_rx_csum = tun_set_rx_csum
 };

+static int tun_init_net(struct net *net)
+{
+ struct tun_net *tn;
+
+ tn = kmalloc(sizeof(*tn), GFP_KERNEL);
+ if (tn == NULL)
+  return -ENOMEM;
+
+ INIT_LIST_HEAD(&tn->dev_list);
+
+ if (net_assign_generic(net, tun_net_id, tn)) {
+  kfree(tn);
+  return -ENOMEM;
+ }
+
+ return 0;
+}
+
+static void tun_exit_net(struct net *net)
+{
+ struct tun_net *tn;
+
+ tn = net_generic(net, tun_net_id);
+ kfree(tn);
+}
+
+static struct pernet_operations tun_net_ops = {
+ .init = tun_init_net,
+ .exit = tun_exit_net,
+};
+
 static int __init tun_init(void)
 {
  int ret = 0;
@@ -880,9 +917,22 @@ static int __init tun_init(void)
```

```
  printk(KERN_INFO "tun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
  printk(KERN_INFO "tun: %s\n", DRV_COPYRIGHT);

+ ret = register_pernet_gen_device(&tun_net_id, &tun_net_ops);
+ if (ret) {
+  printk(KERN_ERR "tun: Can't register pernet ops\n");
+  goto err_pernet;
+ }
+
  ret = misc_register(&tun_miscdev);
- if (ret)
+ if (ret) {
   printk(KERN_ERR "tun: Can't register misc device %d\n", TUN_MINOR);
+  goto err_misc;
+ }
+ return 0;
+
+err_misc:
+ unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
+err_pernet:
  return ret;
 }

@@ -899,6 +949,7 @@ static void tun_cleanup(void)
  }
  rtnl_unlock();

+ unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
 }

 module_init(tun_init);
--
1.5.3.4
```

Subject: [PATCH 4/14][TUN]: Actually make the tun_dev_list per-net.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:48:10 GMT
View Forum Message <> Reply to Message

Remove the static tun_dev_list and replace its occurrences in
driver with per-net one.

It is used in two places - in tun_set_iff and tun_cleanup. In

the first case it's legal to use current net_ns. In the cleanup
call - move the loop, that unregisters all devices in net exit
hook.

This shows how to use the generic pointer.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 drivers/net/tun.c | 37 ++++++++++++++++++++++---------------
 1 files changed, 19 insertions(+), 18 deletions(-)

diff --git a/drivers/net/tun.c b/drivers/net/tun.c
index 9bfba02..74263a4 100644
--- a/drivers/net/tun.c
+++ b/drivers/net/tun.c
@@ -62,6 +62,7 @@
 #include <linux/if_ether.h>
 #include <linux/if_tun.h>
 #include <linux/crc32.h>
+#include <linux/nsproxy.h>
 #include <net/net_namespace.h>
 #include <net/netns/generic.h>

@@ -79,7 +80,6 @@ struct tun_net {
  struct list_head dev_list;
 };

-static LIST_HEAD(tun_dev_list);
 static const struct ethtool_ops tun_ethtool_ops;

 /* Net device open. */
@@ -443,12 +443,12 @@ static void tun_setup(struct net_device *dev)
  dev->destructor = free_netdev;
 }

-static struct tun_struct *tun_get_by_name(const char *name)
+static struct tun_struct *tun_get_by_name(struct tun_net *tn, const char *name)
 {
  struct tun_struct *tun;

  ASSERT_RTNL();
- list_for_each_entry(tun, &tun_dev_list, list) {
+ list_for_each_entry(tun, &tn->dev_list, list) {
   if (!strncmp(tun->dev->name, name, IFNAMSIZ))
       return tun;
  }
@@ -456,13 +456,15 @@ static struct tun_struct *tun_get_by_name(const char *name)

```
  return NULL;
 }

-static int tun_set_iff(struct file *file, struct ifreq *ifr)
+static int tun_set_iff(struct net *net, struct file *file, struct ifreq *ifr)
 {
+ struct tun_net *tn;
  struct tun_struct *tun;
  struct net_device *dev;
  int err;

- tun = tun_get_by_name(ifr->ifr_name);
+ tn = net_generic(net, tun_net_id);
+ tun = tun_get_by_name(tn, ifr->ifr_name);
  if (tun) {
   if (tun->attached)
    return -EBUSY;
@@ -475,7 +477,7 @@ static int tun_set_iff(struct file *file, struct ifreq *ifr)
        !capable(CAP_NET_ADMIN))
    return -EPERM;
  }
- else if (__dev_get_by_name(&init_net, ifr->ifr_name))
+ else if (__dev_get_by_name(net, ifr->ifr_name))
   return -EINVAL;
  else {
   char *name;
@@ -528,7 +530,7 @@ static int tun_set_iff(struct file *file, struct ifreq *ifr)
   if (err < 0)
    goto err_free_dev;

-  list_add(&tun->list, &tun_dev_list);
+  list_add(&tun->list, &tn->dev_list);
  }

  DBG(KERN_INFO "%s: tun_set_iff\n", tun->dev->name);
@@ -573,7 +575,7 @@ static int tun_chr_ioctl(struct inode *inode, struct file *file,
   ifr.ifr_name[IFNAMSIZ-1] = '\0';

   rtnl_lock();
-  err = tun_set_iff(file, &ifr);
+  err = tun_set_iff(current->nsproxy->net_ns, file, &ifr);
   rtnl_unlock();

   if (err)
@@ -900,8 +902,17 @@ static int tun_init_net(struct net *net)
 static void tun_exit_net(struct net *net)
 {
  struct tun_net *tn;
```

```
+ struct tun_struct *tun, *nxt;

  tn = net_generic(net, tun_net_id);
+
+ rtnl_lock();
+ list_for_each_entry_safe(tun, nxt, &tn->dev_list, list) {
+  DBG(KERN_INFO "%s cleaned up\n", tun->dev->name);
+  unregister_netdevice(tun->dev);
+ }
+ rtnl_unlock();
+
  kfree(tn);
 }

@@ -938,17 +949,7 @@ err_pernet:

 static void tun_cleanup(void)
 {
- struct tun_struct *tun, *nxt;
-
  misc_deregister(&tun_miscdev);
-
- rtnl_lock();
- list_for_each_entry_safe(tun, nxt, &tun_dev_list, list) {
-  DBG(KERN_INFO "%s cleaned up\n", tun->dev->name);
-  unregister_netdevice(tun->dev);
- }
- rtnl_unlock();
-
  unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
 }

--
1.5.3.4
```


Subject: [PATCH 5/14][TUN]: Allow to register tun devices in namespace.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:50:09 GMT
View Forum Message <> Reply to Message

This is basically means that a net is set for a new device, but
actually this involves two more steps:

1. mark the tun device as "local", i.e. do not allow for it to
   move across namespaces.

This is done so, since tun device is most often associated to some
file (and thus to some process) and moving the device alone is not
valid while keeping the file and the process outside (and the tun
devices are not always "persistent" ;)).

2. get the tun device's net when tun becomes attached and put one
   when it becomes detached.

This is needed to handle the case when a task owning the tun dies,
but a files lives for some more time - in this case we must not
allow for net to be freed, since its exit hook will spoil that file's
private data by unregistering the tun from under tun_chr_close.

(The TUN virtualization patches were approved by TUN maintainer)

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 drivers/net/tun.c | 4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)

diff --git a/drivers/net/tun.c b/drivers/net/tun.c
index 74263a4..893e92f 100644
--- a/drivers/net/tun.c
+++ b/drivers/net/tun.c
@@ -441,6 +441,7 @@ static void tun_setup(struct net_device *dev)
  dev->stop = tun_net_close;
  dev->ethtool_ops = &tun_ethtool_ops;
  dev->destructor = free_netdev;
+ dev->features |= NETIF_F_NETNS_LOCAL;
 }

 static struct tun_struct *tun_get_by_name(struct tun_net *tn, const char *name)
@@ -508,6 +509,7 @@ static int tun_set_iff(struct net *net, struct file *file, struct ifreq *ifr)
   if (!dev)
    return -ENOMEM;

+ dev_net_set(dev, net);
   tun = netdev_priv(dev);
   tun->dev = dev;
   tun->flags = flags;
@@ -547,6 +549,7 @@ static int tun_set_iff(struct net *net, struct file *file, struct ifreq *ifr)

  file->private_data = tun;

```
  tun->attached = 1;
+ get_net(dev_net(tun->dev));

  strcpy(ifr->ifr_name, tun->dev->name);
  return 0;
@@ -762,6 +765,7 @@ static int tun_chr_close(struct inode *inode, struct file *file)
  /* Detach from net device */
  file->private_data = NULL;
  tun->attached = 0;
+ put_net(dev_net(tun->dev));

  /* Drop read queue */
  skb_queue_purge(&tun->readq);
--
1.5.3.4
```

Subject: [PATCH 6/14][RTNL]: Introduce the rtnl_kill_links call.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:52:33 GMT
View Forum Message <> Reply to Message

... which will kill all the devices in the given net with
the given rtnl_link_ops. Will be used in VLAN patches later.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 include/net/rtnetlink.h |    1 +
 net/core/rtnetlink.c    |   29 ++++++++++++++++++++--------
 2 files changed, 22 insertions(+), 8 deletions(-)

diff --git a/include/net/rtnetlink.h b/include/net/rtnetlink.h
index 793863e..3c1895e 100644
--- a/include/net/rtnetlink.h
+++ b/include/net/rtnetlink.h
@@ -74,6 +74,7 @@ struct rtnl_link_ops {

 extern int __rtnl_link_register(struct rtnl_link_ops *ops);
 extern void __rtnl_link_unregister(struct rtnl_link_ops *ops);
+extern void rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops);

 extern int rtnl_link_register(struct rtnl_link_ops *ops);
 extern void rtnl_link_unregister(struct rtnl_link_ops *ops);
```

```
diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
index da99ac0..bc39e41 100644
--- a/net/core/rtnetlink.c
+++ b/net/core/rtnetlink.c
@@ -269,6 +269,26 @@ int rtnl_link_register(struct rtnl_link_ops *ops)

 EXPORT_SYMBOL_GPL(rtnl_link_register);

+static void __rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
+{
+ struct net_device *dev;
+restart:
+ for_each_netdev(net, dev) {
+  if (dev->rtnl_link_ops == ops) {
+   ops->dellink(dev);
+   goto restart;
+  }
+ }
+}
+
+void rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
+{
+ rtnl_lock();
+ __rtnl_kill_links(net, ops);
+ rtnl_unlock();
+}
+EXPORT_SYMBOL_GPL(rtnl_kill_links);
+
 /**
  * __rtnl_link_unregister - Unregister rtnl_link_ops from rtnetlink.
  * @ops: struct rtnl_link_ops * to unregister
@@ -277,17 +297,10 @@ EXPORT_SYMBOL_GPL(rtnl_link_register);
  */
 void __rtnl_link_unregister(struct rtnl_link_ops *ops)
 {
- struct net_device *dev, *n;
  struct net *net;

  for_each_net(net) {
-restart:
-  for_each_netdev_safe(net, dev, n) {
-   if (dev->rtnl_link_ops == ops) {
-    ops->dellink(dev);
-    goto restart;
-   }
-  }
+  __rtnl_kill_links(net, ops);
  }
```

```
 list_del(&ops->list);
 }
--
1.5.3.4
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 7/14][VLAN]: Tag vlan_group with device, not ifindex.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:54:00 GMT
View Forum Message <> Reply to Message

We'll have to lookup the vlan_group by two keys - ifindex and
net. Turning the vlan_group lookup key to struct net_device
pointer will make this process easier.

Besides, this will eliminate one more place in the networking,
that assumes that indexes are unique in the kernel.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 include/linux/if_vlan.h |    4 +++-
 net/8021q/vlan.c        |   22 ++++++++++++----------
 2 files changed, 14 insertions(+), 12 deletions(-)

diff --git a/include/linux/if_vlan.h b/include/linux/if_vlan.h
index edd55af..15ace02 100644
--- a/include/linux/if_vlan.h
+++ b/include/linux/if_vlan.h
@@ -81,7 +81,9 @@ extern void vlan_ioctl_set(int (*hook)(struct net *, void __user *));
 #define VLAN_GROUP_ARRAY_PART_LEN
(VLAN_GROUP_ARRAY_LEN/VLAN_GROUP_ARRAY_SPLIT_PARTS)

 struct vlan_group {
- int real_dev_ifindex; /* The ifindex of the ethernet(like) device the vlan is attached to. */
+ struct net_device *real_dev; /* The ethernet(like) device
+         * the vlan is attached to.
+         */
   unsigned int  nr_vlans;
   struct hlist_node hlist; /* linked list */
   struct net_device **vlan_devices_arrays[VLAN_GROUP_ARRAY_SPLIT_PARTS];
diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index 5975ec3..cf8d810 100644
--- a/net/8021q/vlan.c
```

```
+++ b/net/8021q/vlan.c
@@ -65,14 +65,14 @@ static inline unsigned int vlan_grp_hashfn(unsigned int idx)
 }

 /* Must be invoked with RCU read lock (no preempt) */
-static struct vlan_group *__vlan_find_group(int real_dev_ifindex)
+static struct vlan_group *__vlan_find_group(struct net_device *real_dev)
 {
  struct vlan_group *grp;
  struct hlist_node *n;
- int hash = vlan_grp_hashfn(real_dev_ifindex);
+ int hash = vlan_grp_hashfn(real_dev->ifindex);

  hlist_for_each_entry_rcu(grp, n, &vlan_group_hash[hash], hlist) {
-  if (grp->real_dev_ifindex == real_dev_ifindex)
+  if (grp->real_dev == real_dev)
    return grp;
  }

@@ -86,7 +86,7 @@ static struct vlan_group *__vlan_find_group(int real_dev_ifindex)
 struct net_device *__find_vlan_dev(struct net_device *real_dev,
        unsigned short VID)
 {
- struct vlan_group *grp = __vlan_find_group(real_dev->ifindex);
+ struct vlan_group *grp = __vlan_find_group(real_dev);

  if (grp)
   return vlan_group_get_device(grp, VID);
@@ -103,7 +103,7 @@ static void vlan_group_free(struct vlan_group *grp)
  kfree(grp);
 }

-static struct vlan_group *vlan_group_alloc(int ifindex)
+static struct vlan_group *vlan_group_alloc(struct net_device *real_dev)
 {
  struct vlan_group *grp;

@@ -111,9 +111,9 @@ static struct vlan_group *vlan_group_alloc(int ifindex)
  if (!grp)
   return NULL;

- grp->real_dev_ifindex = ifindex;
+ grp->real_dev = real_dev;
  hlist_add_head_rcu(&grp->hlist,
-      &vlan_group_hash[vlan_grp_hashfn(ifindex)]);
+   &vlan_group_hash[vlan_grp_hashfn(real_dev->ifindex)]);
  return grp;
 }
```

```
@@ -151,7 +151,7 @@ void unregister_vlan_dev(struct net_device *dev)

  ASSERT_RTNL();

- grp = __vlan_find_group(real_dev->ifindex);
+ grp = __vlan_find_group(real_dev);
  BUG_ON(!grp);

  vlan_proc_rem_dev(dev);
@@ -246,9 +246,9 @@ int register_vlan_dev(struct net_device *dev)
  struct vlan_group *grp, *ngrp = NULL;
  int err;

- grp = __vlan_find_group(real_dev->ifindex);
+ grp = __vlan_find_group(real_dev);
  if (!grp) {
-  ngrp = grp = vlan_group_alloc(real_dev->ifindex);
+  ngrp = grp = vlan_group_alloc(real_dev);
   if (!grp)
    return -ENOBUFS;
  }
@@ -412,7 +412,7 @@ static int vlan_device_event(struct notifier_block *unused, unsigned long
event,
   goto out;
  }

- grp = __vlan_find_group(dev->ifindex);
+ grp = __vlan_find_group(dev);
  if (!grp)
   goto out;

--
1.5.3.4
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 8/14][VLAN]: Introduce the vlan_net structure.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:54:55 GMT
View Forum Message <> Reply to Message

It is empty now, but it will be populated later.
And it uses net generic pointers like TUN does.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 net/8021q/vlan.c | 46 ++
 net/8021q/vlan.h | 5 +++++
 2 files changed, 51 insertions(+), 0 deletions(-)

```
diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index cf8d810..9296601 100644
--- a/net/8021q/vlan.c
+++ b/net/8021q/vlan.c
@@ -32,6 +32,7 @@
 #include <linux/rtnetlink.h>
 #include <linux/notifier.h>
 #include <net/net_namespace.h>
+#include <net/netns/generic.h>

 #include <linux/if_vlan.h>
 #include "vlan.h"
@@ -41,6 +42,8 @@

 /* Global VLAN variables */

+int vlan_net_id;
+
 /* Our listing of VLAN group(s) */
 static struct hlist_head vlan_group_hash[VLAN_GRP_HASH_SIZE];

@@ -625,6 +628,41 @@ out:
  return err;
 }

+static int vlan_init_net(struct net *net)
+{
+ int err;
+ struct vlan_net *vn;
+
+ err = -ENOMEM;
+ vn = kzalloc(sizeof(struct vlan_net), GFP_KERNEL);
+ if (vn == NULL)
+  goto err_alloc;
+
+ err = net_assign_generic(net, vlan_net_id, vn);
+ if (err < 0)
+  goto err_assign;
+
+ return 0;
+
```

```
+err_assign:
+ kfree(vn);
+err_alloc:
+ return err;
+}
+
+static void vlan_exit_net(struct net *net)
+{
+ struct vlan_net *vn;
+
+ vn = net_generic(net, vlan_net_id);
+ kfree(vn);
+}
+
+static struct pernet_operations vlan_net_ops = {
+ .init = vlan_init_net,
+ .exit = vlan_exit_net,
+};
+
 static int __init vlan_proto_init(void)
 {
  int err;
@@ -632,6 +670,10 @@ static int __init vlan_proto_init(void)
  pr_info("%s v%s %s\n", vlan_fullname, vlan_version, vlan_copyright);
  pr_info("All bugs added by %s\n", vlan_buggyright);

+ err = register_pernet_gen_device(&vlan_net_id, &vlan_net_ops);
+ if (err < 0)
+  goto err0;
+
  err = vlan_proc_init();
  if (err < 0)
   goto err1;
@@ -653,6 +695,8 @@ err3:
 err2:
  vlan_proc_cleanup();
 err1:
+ unregister_pernet_gen_device(vlan_net_id, &vlan_net_ops);
+err0:
  return err;
 }

@@ -673,6 +717,8 @@ static void __exit vlan_cleanup_module(void)

  vlan_proc_cleanup();

+ unregister_pernet_gen_device(vlan_net_id, &vlan_net_ops);
+
```

```
 synchronize_net();
 }

diff --git a/net/8021q/vlan.h b/net/8021q/vlan.h
index 51271ae..f27d8d6 100644
--- a/net/8021q/vlan.h
+++ b/net/8021q/vlan.h
@@ -50,4 +50,9 @@ static inline int is_vlan_dev(struct net_device *dev)
  return dev->priv_flags & IFF_802_1Q_VLAN;
 }

+extern int vlan_net_id;
+
+struct vlan_net {
+};
+
 #endif /* !(__BEN_VLAN_802_1Q_INC__) */
--
1.5.3.4
```

Subject: [PATCH 9/14][VLAN]: Add net argument to proc init/cleanup calls.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:55:34 GMT
View Forum Message <> Reply to Message

Proc files will be created in each net, so prepare them for this
change just to make patches smaller.

The net != &init_net checks in them are for git-bisect sanity, but
I'll drop them soon.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 net/8021q/vlan.c     |   15 +++++++--------
 net/8021q/vlanproc.c |   12 ++++++++---
 net/8021q/vlanproc.h |   10 ++++++----
 3 files changed, 22 insertions(+), 15 deletions(-)

diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index 9296601..541542e 100644
--- a/net/8021q/vlan.c
+++ b/net/8021q/vlan.c
```

```
@@ -642,8 +642,14 @@ static int vlan_init_net(struct net *net)
  if (err < 0)
   goto err_assign;

+ err = vlan_proc_init(net);
+ if (err < 0)
+  goto err_proc;
+
  return 0;

+err_proc:
+ /* nothing */
 err_assign:
  kfree(vn);
 err_alloc:
@@ -655,6 +661,7 @@ static void vlan_exit_net(struct net *net)
  struct vlan_net *vn;

  vn = net_generic(net, vlan_net_id);
+ vlan_proc_cleanup(net);
  kfree(vn);
 }

@@ -674,10 +681,6 @@ static int __init vlan_proto_init(void)
  if (err < 0)
   goto err0;

- err = vlan_proc_init();
- if (err < 0)
-  goto err1;
-
  err = register_netdevice_notifier(&vlan_notifier_block);
  if (err < 0)
   goto err2;
@@ -693,8 +696,6 @@ static int __init vlan_proto_init(void)
 err3:
  unregister_netdevice_notifier(&vlan_notifier_block);
 err2:
- vlan_proc_cleanup();
-err1:
  unregister_pernet_gen_device(vlan_net_id, &vlan_net_ops);
 err0:
  return err;
@@ -715,8 +716,6 @@ static void __exit vlan_cleanup_module(void)
  for (i = 0; i < VLAN_GRP_HASH_SIZE; i++)
   BUG_ON(!hlist_empty(&vlan_group_hash[i]));

- vlan_proc_cleanup();
```

```
-
  unregister_pernet_gen_device(vlan_net_id, &vlan_net_ops);

  synchronize_net();
diff --git a/net/8021q/vlanproc.c b/net/8021q/vlanproc.c
index 24cd96e..4d13aeb 100644
--- a/net/8021q/vlanproc.c
+++ b/net/8021q/vlanproc.c
@@ -138,8 +138,11 @@ static const char *vlan_name_type_str[VLAN_NAME_TYPE_HIGHEST]
= {
  * Clean up /proc/net/vlan entries
  */

-void vlan_proc_cleanup(void)
+void vlan_proc_cleanup(struct net *net)
 {
+ if (net != &init_net)
+  return;
+
  if (proc_vlan_conf)
   remove_proc_entry(name_conf, proc_vlan_dir);

@@ -155,8 +158,11 @@ void vlan_proc_cleanup(void)
  * Create /proc/net/vlan entries
  */

-int __init vlan_proc_init(void)
+int vlan_proc_init(struct net *net)
 {
+ if (net != &init_net)
+  return 0;
+
  proc_vlan_dir = proc_mkdir(name_root, init_net.proc_net);
  if (!proc_vlan_dir)
   goto err;
@@ -169,7 +175,7 @@ int __init vlan_proc_init(void)

 err:
  pr_err("%s: can't create entry in proc filesystem!\n", __func__);
- vlan_proc_cleanup();
+ vlan_proc_cleanup(net);
  return -ENOBUFS;
 }

diff --git a/net/8021q/vlanproc.h b/net/8021q/vlanproc.h
index da542ca..063f60a 100644
--- a/net/8021q/vlanproc.h
+++ b/net/8021q/vlanproc.h
```

```
@@ -2,15 +2,17 @@
 #define __BEN_VLAN_PROC_INC__

 #ifdef CONFIG_PROC_FS
-int vlan_proc_init(void);
+struct net;
+
+int vlan_proc_init(struct net *net);
 int vlan_proc_rem_dev(struct net_device *vlandev);
 int vlan_proc_add_dev(struct net_device *vlandev);
-void vlan_proc_cleanup(void);
+void vlan_proc_cleanup(struct net *net);

 #else /* No CONFIG_PROC_FS */

-#define vlan_proc_init() (0)
-#define vlan_proc_cleanup() do {} while (0)
+#define vlan_proc_init(net) (0)
+#define vlan_proc_cleanup(net) do {} while (0)
 #define vlan_proc_add_dev(dev) ({(void)(dev), 0; })
 #define vlan_proc_rem_dev(dev) ({(void)(dev), 0; })
 #endif
--
1.5.3.4
```

Subject: [PATCH 10/14][VLAN]: Create proc files in proper net.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:55:44 GMT

View Forum Message <> Reply to Message

This is essentially a PATCH #9 part 2 - use the net
pointer passed in.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 net/8021q/vlan.h     |    6 ++++++
 net/8021q/vlanproc.c |   44 +++++++++++++++++-------------------------
 2 files changed, 23 insertions(+), 27 deletions(-)

diff --git a/net/8021q/vlan.h b/net/8021q/vlan.h
index f27d8d6..7258357 100644
--- a/net/8021q/vlan.h
```

```
+++ b/net/8021q/vlan.h
@@ -52,7 +52,13 @@ static inline int is_vlan_dev(struct net_device *dev)

 extern int vlan_net_id;

+struct proc_dir_entry;
+
 struct vlan_net {
+ /* /proc/net/vlan */
+ struct proc_dir_entry *proc_vlan_dir;
+ /* /proc/net/vlan/config */
+ struct proc_dir_entry *proc_vlan_conf;
 };

 #endif /* !(__BEN_VLAN_802_1Q_INC__) */
diff --git a/net/8021q/vlanproc.c b/net/8021q/vlanproc.c
index 4d13aeb..995544b 100644
--- a/net/8021q/vlanproc.c
+++ b/net/8021q/vlanproc.c
@@ -34,6 +34,7 @@
 #include <linux/netdevice.h>
 #include <linux/if_vlan.h>
 #include <net/net_namespace.h>
+#include <net/netns/generic.h>
 #include "vlanproc.h"
 #include "vlan.h"

@@ -111,18 +112,6 @@ static const struct file_operations vlandev_fops = {
  * Proc filesystem derectory entries.
  */

-/*
- * /proc/net/vlan
- */
-
-static struct proc_dir_entry *proc_vlan_dir;
-
-/*
- * /proc/net/vlan/config
- */
-
-static struct proc_dir_entry *proc_vlan_conf;
-
 /* Strings */
 static const char *vlan_name_type_str[VLAN_NAME_TYPE_HIGHEST] = {
     [VLAN_NAME_TYPE_RAW_PLUS_VID]        = "VLAN_NAME_TYPE_RAW_PLUS_VID",
@@ -140,14 +129,13 @@ static const char
*vlan_name_type_str[VLAN_NAME_TYPE_HIGHEST] = {
```

```
 void vlan_proc_cleanup(struct net *net)
 {
- if (net != &init_net)
-  return;
+ struct vlan_net *vn = net_generic(net, vlan_net_id);

- if (proc_vlan_conf)
-  remove_proc_entry(name_conf, proc_vlan_dir);
+ if (vn->proc_vlan_conf)
+  remove_proc_entry(name_conf, vn->proc_vlan_dir);

- if (proc_vlan_dir)
-  proc_net_remove(&init_net, name_root);
+ if (vn->proc_vlan_dir)
+  proc_net_remove(net, name_root);

  /* Dynamically added entries should be cleaned up as their vlan_device
   * is removed, so we should not have to take care of it here...
@@ -160,16 +148,15 @@ void vlan_proc_cleanup(struct net *net)

 int vlan_proc_init(struct net *net)
 {
- if (net != &init_net)
-  return 0;
+ struct vlan_net *vn = net_generic(net, vlan_net_id);

- proc_vlan_dir = proc_mkdir(name_root, init_net.proc_net);
- if (!proc_vlan_dir)
+ vn->proc_vlan_dir = proc_net_mkdir(net, name_root, net->proc_net);
+ if (!vn->proc_vlan_dir)
   goto err;

- proc_vlan_conf = proc_create(name_conf, S_IFREG|S_IRUSR|S_IWUSR,
-         proc_vlan_dir, &vlan_fops);
- if (!proc_vlan_conf)
+ vn->proc_vlan_conf = proc_create(name_conf, S_IFREG|S_IRUSR|S_IWUSR,
+         vn->proc_vlan_dir, &vlan_fops);
+ if (!vn->proc_vlan_conf)
   goto err;
  return 0;

@@ -186,9 +173,10 @@ err:
 int vlan_proc_add_dev(struct net_device *vlandev)
 {
  struct vlan_dev_info *dev_info = vlan_dev_info(vlandev);
+ struct vlan_net *vn = net_generic(dev_net(vlandev), vlan_net_id);
```

```
  dev_info->dent = proc_create(vlandev->name, S_IFREG|S_IRUSR|S_IWUSR,
-         proc_vlan_dir, &vlandev_fops);
+         vn->proc_vlan_dir, &vlandev_fops);
  if (!dev_info->dent)
   return -ENOBUFS;

@@ -201,10 +189,12 @@ int vlan_proc_add_dev(struct net_device *vlandev)
  */
 int vlan_proc_rem_dev(struct net_device *vlandev)
 {
+ struct vlan_net *vn = net_generic(dev_net(vlandev), vlan_net_id);
+
  /** NOTE:  This will consume the memory pointed to by dent, it seems. */
  if (vlan_dev_info(vlandev)->dent) {
   remove_proc_entry(vlan_dev_info(vlandev)->dent->name,
-      proc_vlan_dir);
+      vn->proc_vlan_dir);
   vlan_dev_info(vlandev)->dent = NULL;
  }
  return 0;
--
1.5.3.4
```

Subject: [PATCH 11/14][VLAN]: Make /proc/net/vlan/conf file show per-net info.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:57:06 GMT
View Forum Message <> Reply to Message

And this one if PATCH #9 part 3 :)
After this proc may show relevant to each net namespace info.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 net/8021q/vlanproc.c |   13 ++++++++-----
 1 files changed, 8 insertions(+), 5 deletions(-)

diff --git a/net/8021q/vlanproc.c b/net/8021q/vlanproc.c
index 995544b..cc17b72 100644
--- a/net/8021q/vlanproc.c
+++ b/net/8021q/vlanproc.c
@@ -80,7 +80,8 @@ static const struct seq_operations vlan_seq_ops = {
```

```
 static int vlan_seq_open(struct inode *inode, struct file *file)
 {
- return seq_open(file, &vlan_seq_ops);
+ return seq_open_net(inode, file, &vlan_seq_ops,
+    sizeof(struct seq_net_private));
 }

 static const struct file_operations vlan_fops = {
@@ -88,7 +89,7 @@ static const struct file_operations vlan_fops = {
  .open    = vlan_seq_open,
  .read    = seq_read,
  .llseek  = seq_lseek,
- .release = seq_release,
+ .release = seq_release_net,
 };

 /*
@@ -211,6 +212,7 @@ static void *vlan_seq_start(struct seq_file *seq, loff_t *pos)
  __acquires(dev_base_lock)
 {
  struct net_device *dev;
+ struct net *net = seq_file_net(seq);
  loff_t i = 1;

  read_lock(&dev_base_lock);
@@ -218,7 +220,7 @@ static void *vlan_seq_start(struct seq_file *seq, loff_t *pos)
  if (*pos == 0)
   return SEQ_START_TOKEN;

- for_each_netdev(&init_net, dev) {
+ for_each_netdev(net, dev) {
   if (!is_vlan_dev(dev))
    continue;

@@ -232,14 +234,15 @@ static void *vlan_seq_start(struct seq_file *seq, loff_t *pos)
 static void *vlan_seq_next(struct seq_file *seq, void *v, loff_t *pos)
 {
  struct net_device *dev;
+ struct net *net = seq_file_net(seq);

  ++*pos;

  dev = (struct net_device *)v;
  if (v == SEQ_START_TOKEN)
-  dev = net_device_entry(&init_net.dev_base_head);
+  dev = net_device_entry(&net->dev_base_head);

- for_each_netdev_continue(&init_net, dev) {
```

```
+ for_each_netdev_continue(net, dev) {
   if (!is_vlan_dev(dev))
    continue;

--
1.5.3.4
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 12/14][VLAN]: Make vlan_name_type per-net.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:58:19 GMT
View Forum Message <> Reply to Message

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---
 net/8021q/vlan.c     |   14 +++++++++-----
 net/8021q/vlan.h     |    4 ++--
 net/8021q/vlanproc.c |    7 +++++--
 3 files changed, 16 insertions(+), 9 deletions(-)

diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index 541542e..5cacad0 100644
--- a/net/8021q/vlan.c
+++ b/net/8021q/vlan.c
@@ -52,9 +52,6 @@ static char vlan_version[] = DRV_VERSION;
 static char vlan_copyright[] = "Ben Greear <greearb@candelatech.com>";
 static char vlan_buggyright[] = "David S. Miller <davem@redhat.com>";

-/* Determines interface naming scheme. */
-unsigned short vlan_name_type = VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD;
-
 static struct packet_type vlan_packet_type = {
  .type = __constant_htons(ETH_P_8021Q),
  .func = vlan_skb_recv, /* VLAN receive method */
@@ -299,6 +296,8 @@ static int register_vlan_device(struct net_device *real_dev,
     unsigned short VLAN_ID)
 {
  struct net_device *new_dev;
+ struct net *net = dev_net(real_dev);
+ struct vlan_net *vn = net_generic(net, vlan_net_id);
  char name[IFNAMSIZ];
  int err;
```

```
@@ -310,7 +309,7 @@ static int register_vlan_device(struct net_device *real_dev,
   return err;

  /* Gotta set up the fields for the device. */
- switch (vlan_name_type) {
+ switch (vn->name_type) {
  case VLAN_NAME_TYPE_RAW_PLUS_VID:
   /* name will look like:  eth1.0005 */
   snprintf(name, IFNAMSIZ, "%s.%.4i", real_dev->name, VLAN_ID);
@@ -580,7 +579,10 @@ static int vlan_ioctl_handler(struct net *net, void __user *arg)
    break;
   if ((args.u.name_type >= 0) &&
       (args.u.name_type < VLAN_NAME_TYPE_HIGHEST)) {
-   vlan_name_type = args.u.name_type;
+   struct vlan_net *vn;
+
+   vn = net_generic(net, vlan_net_id);
+   vn->name_type = args.u.name_type;
    err = 0;
   } else {
    err = -EINVAL;
@@ -642,6 +644,8 @@ static int vlan_init_net(struct net *net)
  if (err < 0)
   goto err_assign;

+ vn->name_type = VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD;
+
  err = vlan_proc_init(net);
  if (err < 0)
   goto err_proc;
diff --git a/net/8021q/vlan.h b/net/8021q/vlan.h
index 7258357..5229a72 100644
--- a/net/8021q/vlan.h
+++ b/net/8021q/vlan.h
@@ -3,8 +3,6 @@

 #include <linux/if_vlan.h>

-extern unsigned short vlan_name_type;
-
 #define VLAN_GRP_HASH_SHIFT 5
 #define VLAN_GRP_HASH_SIZE (1 << VLAN_GRP_HASH_SHIFT)
 #define VLAN_GRP_HASH_MASK (VLAN_GRP_HASH_SIZE - 1)
@@ -59,6 +57,8 @@ struct vlan_net {
  struct proc_dir_entry *proc_vlan_dir;
  /* /proc/net/vlan/config */
  struct proc_dir_entry *proc_vlan_conf;
+ /* Determines interface naming scheme. */
```

```
+ unsigned short name_type;
 };

 #endif /* !(__BEN_VLAN_802_1Q_INC__) */
diff --git a/net/8021q/vlanproc.c b/net/8021q/vlanproc.c
index cc17b72..daad006 100644
--- a/net/8021q/vlanproc.c
+++ b/net/8021q/vlanproc.c
@@ -260,13 +260,16 @@ static void vlan_seq_stop(struct seq_file *seq, void *v)

 static int vlan_seq_show(struct seq_file *seq, void *v)
 {
+ struct net *net = seq_file_net(seq);
+ struct vlan_net *vn = net_generic(net, vlan_net_id);
+
  if (v == SEQ_START_TOKEN) {
   const char *nmtype = NULL;

   seq_puts(seq, "VLAN Dev name  | VLAN ID\n");

-  if (vlan_name_type < ARRAY_SIZE(vlan_name_type_str))
-      nmtype =  vlan_name_type_str[vlan_name_type];
+  if (vn->name_type < ARRAY_SIZE(vlan_name_type_str))
+      nmtype =  vlan_name_type_str[vn->name_type];

   seq_printf(seq, "Name-Type: %s\n",
       nmtype ? nmtype :  "UNKNOWN");
--
1.5.3.4
```

Subject: [PATCH 13/14][VLAN]: Allows vlan devices registration in net namespace.
Posted by Pavel Emelianov on Thu, 10 Apr 2008 14:59:32 GMT
View Forum Message <> Reply to Message

This one is similar to what I've done for TUN - set the proper
net after device allocation and clean VLANs on net exit.

Plus, drop explicit init_net usage.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
 net/8021q/vlan.c         |    7 ++++----
 net/8021q/vlan_dev.c     |    3 ---
 net/8021q/vlan_netlink.c |    2 +-
 3 files changed, 4 insertions(+), 8 deletions(-)

diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index 5cacad0..7e9d22e 100644
--- a/net/8021q/vlan.c
+++ b/net/8021q/vlan.c
@@ -340,6 +340,7 @@ static int register_vlan_device(struct net_device *real_dev,
  if (new_dev == NULL)
   return -ENOBUFS;

+ dev_net_set(new_dev, net);
  /* need 4 bytes for extra VLAN header info,
   * hope the underlying device can handle it.
   */
@@ -406,9 +407,6 @@ static int vlan_device_event(struct notifier_block *unused, unsigned long
event,
  int i, flgs;
  struct net_device *vlandev;

- if (dev_net(dev) != &init_net)
-  return NOTIFY_DONE;
-
  if (is_vlan_dev(dev)) {
   __vlan_device_event(dev, event);
   goto out;
@@ -534,7 +532,7 @@ static int vlan_ioctl_handler(struct net *net, void __user *arg)
  case GET_VLAN_REALDEV_NAME_CMD:
  case GET_VLAN_VID_CMD:
   err = -ENODEV;
-  dev = __dev_get_by_name(&init_net, args.device1);
+  dev = __dev_get_by_name(net, args.device1);
   if (!dev)
    goto out;

@@ -665,6 +663,7 @@ static void vlan_exit_net(struct net *net)
  struct vlan_net *vn;

  vn = net_generic(net, vlan_net_id);
+ rtnl_kill_links(net, &vlan_link_ops);
  vlan_proc_cleanup(net);
  kfree(vn);
 }
diff --git a/net/8021q/vlan_dev.c b/net/8021q/vlan_dev.c
index 0e3b2d3..9fa6b90 100644
--- a/net/8021q/vlan_dev.c
```

```
+++ b/net/8021q/vlan_dev.c
@@ -153,9 +153,6 @@ int vlan_skb_recv(struct sk_buff *skb, struct net_device *dev,
  struct net_device_stats *stats;
  unsigned short vlan_TCI;

- if (dev_net(dev) != &init_net)
-  goto err_free;
-
  skb = skb_share_check(skb, GFP_ATOMIC);
  if (skb == NULL)
   goto err_free;
diff --git a/net/8021q/vlan_netlink.c b/net/8021q/vlan_netlink.c
index e32eeb3..c93e69e 100644
--- a/net/8021q/vlan_netlink.c
+++ b/net/8021q/vlan_netlink.c
@@ -113,7 +113,7 @@ static int vlan_newlink(struct net_device *dev,

  if (!tb[IFLA_LINK])
   return -EINVAL;
- real_dev = __dev_get_by_index(&init_net, nla_get_u32(tb[IFLA_LINK]));
+ real_dev = __dev_get_by_index(dev_net(dev), nla_get_u32(tb[IFLA_LINK]));
  if (!real_dev)
   return -ENODEV;

--
1.5.3.4
```

Subject: [PATCH 14/14][VLAN]: Migrate proc files when vlan device is moved to namespace.

Posted by Pavel Emelianov on Thu, 10 Apr 2008 15:02:07 GMT

View Forum Message <> Reply to Message

The preparations to this are done with my patch, that fixed
proc on vlan device rename.

This makes /proc/net/vlan/<device> file migrate from one
proc tree to another...

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 net/8021q/vlan.c | 13 ++++++++-----

```
 1 files changed, 8 insertions(+), 5 deletions(-)

diff --git a/net/8021q/vlan.c b/net/8021q/vlan.c
index 7e9d22e..2a739ad 100644
--- a/net/8021q/vlan.c
+++ b/net/8021q/vlan.c
@@ -154,8 +154,6 @@ void unregister_vlan_dev(struct net_device *dev)
  grp = __vlan_find_group(real_dev);
  BUG_ON(!grp);

- vlan_proc_rem_dev(dev);
-
  /* Take it out of our own structures, but be sure to interlock with
   * HW accelerating devices or SW vlan input packet processing.
   */
@@ -278,9 +276,6 @@ int register_vlan_dev(struct net_device *dev)
  if (real_dev->features & NETIF_F_HW_VLAN_FILTER)
   real_dev->vlan_rx_add_vid(real_dev, vlan_id);

- if (vlan_proc_add_dev(dev) < 0)
-  pr_warning("8021q: failed to add proc entry for %s\n",
-      dev->name);
  return 0;

 out_free_group:
@@ -396,6 +391,14 @@ static void __vlan_device_event(struct net_device *dev, unsigned long
event)
    pr_warning("8021q: failed to change proc name for %s\n",
      dev->name);
   break;
+ case NETDEV_REGISTER:
+  if (vlan_proc_add_dev(dev) < 0)
+   pr_warning("8021q: failed to add proc entry for %s\n",
+      dev->name);
+  break;
+ case NETDEV_UNREGISTER:
+  vlan_proc_rem_dev(dev);
+  break;
  }
 }

--
1.5.3.4
```

Subject: Re: [PATCH 1/14][NETNS]: Introduce the net-subsys id generator.
Posted by Paul Menage on Thu, 10 Apr 2008 17:37:53 GMT
View Forum Message <> Reply to Message

On Thu, Apr 10, 2008 at 8:01 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> +int register_pernet_gen_device(int *id, struct pernet_operations *ops)
> +{
> + int error;
> + mutex_lock(&net_mutex);
> +again:
> + error = ida_get_new_above(&net_generic_ids, 1, id);
> + if (error) {
> + if (error == -EAGAIN) {
> + ida_pre_get(&net_generic_ids, GFP_KERNEL);
> + goto again;
> + }

Shouldn't you handle non-EAGAIN errors here?

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 1/14][NETNS]: Introduce the net-subsys id generator.
Posted by Daniel Lezcano on Thu, 10 Apr 2008 20:52:44 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> To make some per-net generic pointers, we need some way to
> address them, i.e. - IDs. This is simple IDA-based IDs
> generator for pernet subsystems. They will be used in the
> next patches.
>
> Since it will be used by devices only (tun and vlan), I make
> it resemble the register_pernet_device functionality.
>
> The new ids is stored in the *id pointer _before_ calling the
> init callback to make this id available in this callback.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> include/net/net_namespace.h | 2 ++
> net/core/net_namespace.c | 36 ++++++++++++++++++++++++++++++++++++
> 2 files changed, 38 insertions(+), 0 deletions(-)

>
> diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
> index 0ab62ed..6971fdb 100644
> --- a/include/net/net_namespace.h
> +++ b/include/net/net_namespace.h
> @@ -181,6 +181,8 @@ extern int register_pernet_subsys(struct pernet_operations *);
> extern void unregister_pernet_subsys(struct pernet_operations *);
> extern int register_pernet_device(struct pernet_operations *);
> extern void unregister_pernet_device(struct pernet_operations *);
> +extern int register_pernet_gen_device(int *id, struct pernet_operations *);
> +extern void unregister_pernet_gen_device(int id, struct pernet_operations *);
>
> struct ctl_path;
> struct ctl_table;
> diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
> index 7b66083..7ef3bac 100644
> --- a/net/core/net_namespace.c
> +++ b/net/core/net_namespace.c
> @@ -5,6 +5,7 @@
> #include <linux/list.h>
> #include <linux/delay.h>
> #include <linux/sched.h>
> +#include <linux/idr.h>
> #include <net/net_namespace.h>
>
> /*
> @@ -253,6 +254,8 @@ static void unregister_pernet_operations(struct pernet_operations
*ops)
> }
> #endif
>
> +static DEFINE_IDA(net_generic_ids);
> +
> /**
>   *   register_pernet_subsys - register a network namespace subsystem
>   * @ops: pernet operations structure for the subsystem
> @@ -330,6 +333,28 @@ int register_pernet_device(struct pernet_operations *ops)
> }
> EXPORT_SYMBOL_GPL(register_pernet_device);
>
> +int register_pernet_gen_device(int *id, struct pernet_operations *ops)
> +{
> + int error;
> + mutex_lock(&net_mutex);
> +again:
> + error = ida_get_new_above(&net_generic_ids, 1, id);
> + if (error) {
> + if (error == -EAGAIN) {

> +   ida_pre_get(&net_generic_ids, GFP_KERNEL);
> +   goto again;
> + }

  goto out;

> + }
> + error = register_pernet_operations(first_device, ops);
> + if (error)
> + ida_remove(&net_generic_ids, *id);
> + else if (first_device == &pernet_list)
> + first_device = &ops->list;

out:

> + mutex_unlock(&net_mutex);
> + return error;
> +}
> +EXPORT_SYMBOL_GPL(register_pernet_gen_device);
> +
> /**
>   *   unregister_pernet_device - unregister a network namespace netdevice
>   * @ops: pernet operations structure to manipulate
> @@ -348,3 +373,14 @@ void unregister_pernet_device(struct pernet_operations *ops)
>   mutex_unlock(&net_mutex);
> }
> EXPORT_SYMBOL_GPL(unregister_pernet_device);
> +
> +void unregister_pernet_gen_device(int id, struct pernet_operations *ops)
> +{
> + mutex_lock(&net_mutex);
> + if (&ops->list == first_device)
> + first_device = first_device->next;
> + unregister_pernet_operations(ops);
> + ida_remove(&net_generic_ids, id);
> + mutex_unlock(&net_mutex);
> +}
> +EXPORT_SYMBOL_GPL(unregister_pernet_gen_device);

--

Sauf indication contraire ci-dessus:

Compagnie IBM France

Courbevoie
RCS Nanterre 552 118 465
Forme Sociale : S.A.S.
Capital Social : 542.737.118 ?
SIREN/SIRET : 552 118 465 02430

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 6/14][RTNL]: Introduce the rtnl_kill_links call.
Posted by Daniel Hokka Zakrisso on Thu, 10 Apr 2008 22:09:55 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> ... which will kill all the devices in the given net with
> the given rtnl_link_ops. Will be used in VLAN patches later.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> include/net/rtnetlink.h | 1 +
> net/core/rtnetlink.c | 29 ++++++++++++++++++++--------
> 2 files changed, 22 insertions(+), 8 deletions(-)
>
> diff --git a/include/net/rtnetlink.h b/include/net/rtnetlink.h
> index 793863e..3c1895e 100644
> --- a/include/net/rtnetlink.h
> +++ b/include/net/rtnetlink.h
> @@ -74,6 +74,7 @@ struct rtnl_link_ops {
>
> extern int __rtnl_link_register(struct rtnl_link_ops *ops);
> extern void __rtnl_link_unregister(struct rtnl_link_ops *ops);
> +extern void rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops);
>
> extern int rtnl_link_register(struct rtnl_link_ops *ops);
> extern void rtnl_link_unregister(struct rtnl_link_ops *ops);
> diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
> index da99ac0..bc39e41 100644
> --- a/net/core/rtnetlink.c
> +++ b/net/core/rtnetlink.c
> @@ -269,6 +269,26 @@ int rtnl_link_register(struct rtnl_link_ops *ops)
>
> EXPORT_SYMBOL_GPL(rtnl_link_register);
>

```
> +static void __rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
> +{
> + struct net_device *dev;
> +restart:
> + for_each_netdev(net, dev) {
> +  if (dev->rtnl_link_ops == ops) {
> +   ops->dellink(dev);
> +   goto restart;
> +  }
> + }
> +}
> +
> +void rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
> +{
> + rtnl_lock();
> + __rtnl_kill_links(net, ops);
> + rtnl_unlock();
> +}
> +EXPORT_SYMBOL_GPL(rtnl_kill_links);
> +
>  /**
>   * __rtnl_link_unregister - Unregister rtnl_link_ops from rtnetlink.
>   * @ops: struct rtnl_link_ops * to unregister
> @@ -277,17 +297,10 @@ EXPORT_SYMBOL_GPL(rtnl_link_register);
>   */
>  void __rtnl_link_unregister(struct rtnl_link_ops *ops)
>  {
> - struct net_device *dev, *n;
>   struct net *net;
>
>   for_each_net(net) {
> -restart:
> -  for_each_netdev_safe(net, dev, n) {
> -   if (dev->rtnl_link_ops == ops) {
> -    ops->dellink(dev);
> -    goto restart;
> -   }
> -  }
> +  __rtnl_kill_links(net, ops);
```

This was _safe, and now it's not. Is that intentional?

```
>   }
>   list_del(&ops->list);
>  }
> --
> 1.5.3.4
```

--
Daniel Hokka Zakrisson

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by paulmck on Fri, 11 Apr 2008 01:01:42 GMT
View Forum Message <> Reply to Message

On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:
> This is the first step in making tuntap devices work in net
> namespaces. The structure mentioned is pointed by generic
> net pointer with tun_net_id id, and tun driver fills one on
> its load. It will contain only the tun devices list.
>
> So declare this structure and introduce net init and exit hooks.

OK, I have to ask... What prevents someone else from invoking
net_generic() concurrently with a call to tun_exit_net(), potentially
obtaining a pointer to the structure that tun_exit_net() is about
to kfree()?

Thanx, Paul

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> drivers/net/tun.c | 53 ++-
> 1 files changed, 52 insertions(+), 1 deletions(-)
>
> diff --git a/drivers/net/tun.c b/drivers/net/tun.c
> index 7b816a0..9bfba02 100644
> --- a/drivers/net/tun.c
> +++ b/drivers/net/tun.c
> @@ -63,6 +63,7 @@
> #include <linux/if_tun.h>
> #include <linux/crc32.h>
> #include <net/net_namespace.h>
> +#include <net/netns/generic.h>
>
> #include <asm/system.h>
> #include <asm/uaccess.h>
> @@ -73,6 +74,11 @@ static int debug;
>
> /* Network device part of the driver */

>
> +static unsigned int tun_net_id;
> +struct tun_net {
> + struct list_head dev_list;
> +};
> +
> static LIST_HEAD(tun_dev_list);
> static const struct ethtool_ops tun_ethtool_ops;
>
> @@ -873,6 +879,37 @@ static const struct ethtool_ops tun_ethtool_ops = {
>   .set_rx_csum = tun_set_rx_csum
> };
>
> +static int tun_init_net(struct net *net)
> +{
> + struct tun_net *tn;
> +
> + tn = kmalloc(sizeof(*tn), GFP_KERNEL);
> + if (tn == NULL)
> + return -ENOMEM;
> +
> + INIT_LIST_HEAD(&tn->dev_list);
> +
> + if (net_assign_generic(net, tun_net_id, tn)) {
> + kfree(tn);
> + return -ENOMEM;
> + }
> +
> + return 0;
> +}
> +
> +static void tun_exit_net(struct net *net)
> +{
> + struct tun_net *tn;
> +
> + tn = net_generic(net, tun_net_id);
> + kfree(tn);
> +}
> +
> +static struct pernet_operations tun_net_ops = {
> + .init = tun_init_net,
> + .exit = tun_exit_net,
> +};
> +
> static int __init tun_init(void)
> {
>   int ret = 0;
> @@ -880,9 +917,22 @@ static int __init tun_init(void)

>   printk(KERN_INFO "tun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
>   printk(KERN_INFO "tun: %s\n", DRV_COPYRIGHT);
>
> + ret = register_pernet_gen_device(&tun_net_id, &tun_net_ops);
> + if (ret) {
> + printk(KERN_ERR "tun: Can't register pernet ops\n");
> + goto err_pernet;
> + }
> +
>   ret = misc_register(&tun_miscdev);
> - if (ret)
> + if (ret) {
>   printk(KERN_ERR "tun: Can't register misc device %d\n", TUN_MINOR);
> + goto err_misc;
> + }
> + return 0;
> +
> +err_misc:
> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
> +err_pernet:
>   return ret;
> }
>
> @@ -899,6 +949,7 @@ static void tun_cleanup(void)
>   }
>   rtnl_unlock();
>
> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
> }
>
> module_init(tun_init);
> --
> 1.5.3.4
>
> --
> To unsubscribe from this list: send the line "unsubscribe netdev" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 1/14][NETNS]: Introduce the net-subsys id generator.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 07:34:59 GMT
View Forum Message <> Reply to Message

>> +int register_pernet_gen_device(int *id, struct pernet_operations *ops)
>> +{
>> + int error;
>> + mutex_lock(&net_mutex);
>> +again:
>> + error = ida_get_new_above(&net_generic_ids, 1, id);
>> + if (error) {
>> + if (error == -EAGAIN) {
>> +   ida_pre_get(&net_generic_ids, GFP_KERNEL);
>> +   goto again;
>> + }
>
>   goto out;
>
>> + }
>> + error = register_pernet_operations(first_device, ops);
>> + if (error)
>> + ida_remove(&net_generic_ids, *id);
>> + else if (first_device == &pernet_list)
>> + first_device = &ops->list;
>
> out:

Oops! Thank, will fix.

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 07:36:24 GMT
View Forum Message <> Reply to Message

Paul E. McKenney wrote:
> On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:
>> This is the first step in making tuntap devices work in net
>> namespaces. The structure mentioned is pointed by generic
>> net pointer with tun_net_id id, and tun driver fills one on
>> its load. It will contain only the tun devices list.
>>
>> So declare this structure and introduce net init and exit hooks.
>
> OK, I have to ask... What prevents someone else from invoking
> net_generic() concurrently with a call to tun_exit_net(), potentially
> obtaining a pointer to the structure that tun_exit_net() is about
> to kfree()?

It's the same as if the tun_net was directly pointed by the struct
net. Nobody can grant, that the pointer got by you from the struct
net is not going to become free, unless you provide this security
by yourself.

But if you call net_generic to get some pointer other than tun_net,
then you're fine (due to RCU), providing you play the same rules with
the pointer you're getting.

Maybe I'm missing something in your question, can you provide some
testcase, that you suspect may cause an OOPS?

> Thanx, Paul
>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>> ---
>> drivers/net/tun.c | 53 ++-
>> 1 files changed, 52 insertions(+), 1 deletions(-)
>>
>> diff --git a/drivers/net/tun.c b/drivers/net/tun.c
>> index 7b816a0..9bfba02 100644
>> --- a/drivers/net/tun.c
>> +++ b/drivers/net/tun.c
>> @@ -63,6 +63,7 @@
>> #include <linux/if_tun.h>
>> #include <linux/crc32.h>
>> #include <net/net_namespace.h>
>> +#include <net/netns/generic.h>
>>
>> #include <asm/system.h>
>> #include <asm/uaccess.h>
>> @@ -73,6 +74,11 @@ static int debug;
>>
>> /* Network device part of the driver */
>>
>> +static unsigned int tun_net_id;
>> +struct tun_net {
>> + struct list_head dev_list;
>> +};
>> +
>> static LIST_HEAD(tun_dev_list);
>> static const struct ethtool_ops tun_ethtool_ops;
>>
>> @@ -873,6 +879,37 @@ static const struct ethtool_ops tun_ethtool_ops = {
>> .set_rx_csum = tun_set_rx_csum
>> };
>>

```
>> +static int tun_init_net(struct net *net)
>> +{
>> + struct tun_net *tn;
>> +
>> + tn = kmalloc(sizeof(*tn), GFP_KERNEL);
>> + if (tn == NULL)
>> +  return -ENOMEM;
>> +
>> + INIT_LIST_HEAD(&tn->dev_list);
>> +
>> + if (net_assign_generic(net, tun_net_id, tn)) {
>> +  kfree(tn);
>> +  return -ENOMEM;
>> + }
>> +
>> + return 0;
>> +}
>> +
>> +static void tun_exit_net(struct net *net)
>> +{
>> + struct tun_net *tn;
>> +
>> + tn = net_generic(net, tun_net_id);
>> + kfree(tn);
>> +}
>> +
>> +static struct pernet_operations tun_net_ops = {
>> + .init = tun_init_net,
>> + .exit = tun_exit_net,
>> +};
>> +
>>  static int __init tun_init(void)
>>  {
>>   int ret = 0;
>> @@ -880,9 +917,22 @@ static int __init tun_init(void)
>>   printk(KERN_INFO "tun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
>>   printk(KERN_INFO "tun: %s\n", DRV_COPYRIGHT);
>>
>> + ret = register_pernet_gen_device(&tun_net_id, &tun_net_ops);
>> + if (ret) {
>> +  printk(KERN_ERR "tun: Can't register pernet ops\n");
>> +  goto err_pernet;
>> + }
>> +
>>   ret = misc_register(&tun_miscdev);
>> - if (ret)
>> + if (ret) {
>>    printk(KERN_ERR "tun: Can't register misc device %d\n", TUN_MINOR);
```

>> + goto err_misc;
>> + }
>> + return 0;
>> +
>> +err_misc:
>> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
>> +err_pernet:
>>   return ret;
>> }
>>
>> @@ -899,6 +949,7 @@ static void tun_cleanup(void)
>>   }
>>   rtnl_unlock();
>>
>> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
>> }
>>
>> module_init(tun_init);
>> --
>> 1.5.3.4
>>
>> --
>> To unsubscribe from this list: send the line "unsubscribe netdev" in
>> the body of a message to majordomo@vger.kernel.org
>> More majordomo info at http://vger.kernel.org/majordomo-info.html
> --
> To unsubscribe from this list: send the line "unsubscribe netdev" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
>

Subject: Re: [PATCH 6/14][RTNL]: Introduce the rtnl_kill_links call.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 07:39:33 GMT
View Forum Message <> Reply to Message

>> diff --git a/net/core/rtnetlink.c b/net/core/rtnetlink.c
>> index da99ac0..bc39e41 100644
>> --- a/net/core/rtnetlink.c
>> +++ b/net/core/rtnetlink.c
>> @@ -269,6 +269,26 @@ int rtnl_link_register(struct rtnl_link_ops *ops)
>>
>> EXPORT_SYMBOL_GPL(rtnl_link_register);

```
>>
>> +static void __rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
>> +{
>> + struct net_device *dev;
>> +restart:
>> + for_each_netdev(net, dev) {
>> +  if (dev->rtnl_link_ops == ops) {
>> +   ops->dellink(dev);
>> +   goto restart;
>> +  }
>> + }
>> +}
>> +
>> +void rtnl_kill_links(struct net *net, struct rtnl_link_ops *ops)
>> +{
>> + rtnl_lock();
>> + __rtnl_kill_links(net, ops);
>> + rtnl_unlock();
>> +}
>> +EXPORT_SYMBOL_GPL(rtnl_kill_links);
>> +
>>  /**
>>   * __rtnl_link_unregister - Unregister rtnl_link_ops from rtnetlink.
>>   * @ops: struct rtnl_link_ops * to unregister
>> @@ -277,17 +297,10 @@ EXPORT_SYMBOL_GPL(rtnl_link_register);
>>   */
>>  void __rtnl_link_unregister(struct rtnl_link_ops *ops)
>>  {
>> - struct net_device *dev, *n;
>>   struct net *net;
>>
>>   for_each_net(net) {
>> -restart:
>> -  for_each_netdev_safe(net, dev, n) {
>> -   if (dev->rtnl_link_ops == ops) {
>> -    ops->dellink(dev);
>> -    goto restart;
>> -   }
>> -  }
>> +  __rtnl_kill_links(net, ops);
>
> This was _safe, and now it's not. Is that intentional?

Yup - we goto restart in case we del some link, so there's no need
in _safe iteration.

This goto was added by Partick (commit 68365458 [NET]: rtnl_link:
fix use-after-free) and I suspect he simply forgot to remove the
```

_safe iterator (I put him in Cc to correct me if I'm wrong).

>> }
>> list_del(&ops->list);
>> }
>> --
>> 1.5.3.4
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 6/14][RTNL]: Introduce the rtnl_kill_links call.
Posted by Patrick McHardy on Fri, 11 Apr 2008 12:48:39 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
>>> for_each_net(net) {
>>> -restart:
>>> - for_each_netdev_safe(net, dev, n) {
>>> -   if (dev->rtnl_link_ops == ops) {
>>> -   ops->dellink(dev);
>>> -   goto restart;
>>> -   }
>>> - }
>>> + __rtnl_kill_links(net, ops);
>> This was _safe, and now it's not. Is that intentional?
>
> Yup - we goto restart in case we del some link, so there's no need
> in _safe iteration.
>
> This goto was added by Partick (commit 68365458 [NET]: rtnl_link:
> fix use-after-free) and I suspect he simply forgot to remove the
> _safe iterator (I put him in Cc to correct me if I'm wrong).

No, that was an oversight, it should be safe to remove.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2/14][NETNS]: Generic per-net pointers.

Pavel Emelyanov wrote:
> Add the elastic array of void * pointer to the struct net.
> The access rules are simple:
>
> 1. register the ops with register_pernet_gen_device to get
>   the id of your private pointer
> 2. call net_assign_generic() to put the private data on the
>   struct net (most preferably this should be done in the
>   ->init callback of the ops registered)
> 3. do not change this pointer while the net is alive;
> 4. use the net_generic() to get the pointer.
>
> When adding a new pointer, I copy the old array, replace it
> with a new one and schedule the old for kfree after an RCU
> grace period.
>
> Since the net_generic explores the net->gen array inside rcu
> read section and once set the net->gen->ptr[x] pointer never
> changes, this grants us a safe access to generic pointers.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> include/net/net_namespace.h | 2 +
> include/net/netns/generic.h | 49 ++
> net/core/net_namespace.c | 62 ++
> 3 files changed, 113 insertions(+), 0 deletions(-)
> create mode 100644 include/net/netns/generic.h
>
> diff --git a/include/net/net_namespace.h b/include/net/net_namespace.h
> index 6971fdb..e3d4eb4 100644
> --- a/include/net/net_namespace.h
> +++ b/include/net/net_namespace.h
> @@ -19,6 +19,7 @@ struct proc_dir_entry;
> struct net_device;
> struct sock;
> struct ctl_table_header;
> +struct net_generic;
>
> struct net {
>   atomic_t count; /* To decided when the network
> @@ -57,6 +58,7 @@ struct net {
> #ifdef CONFIG_NETFILTER
>   struct netns_xt xt;
> #endif
> + struct net_generic *gen;

> };
>
>
> diff --git a/include/net/netns/generic.h b/include/net/netns/generic.h
> new file mode 100644
> index 0000000..e8a6d27
> --- /dev/null
> +++ b/include/net/netns/generic.h
> @@ -0,0 +1,49 @@
> +/*
> + * generic net pointers
> + */
> +
> +#ifndef __NET_GENERIC_H__
> +#define __NET_GENERIC_H__
> +
> +#include <linux/rcupdate.h>
> +
> +/*
> + * Generic net pointers are to be used by modules
> + * to put some private stuff on the struct net without
> + * explicit struct net modification
> + *
> + * The rules are simple:
> + * 1. register the ops with register_pernet_gen_device to get
> + *   the id of your private pointer
> + * 2. call net_assign_generic() to put the private data on the
> + *   struct net (most preferably this should be done in the
> + *   ->init callback of the ops registered)
> + * 3. do not change this pointer while the net is alive.
> + *
> + * After accomplishing all of the above, the private pointer
> + * can be accessed with the net_generic() call.
> + */
> +
> +struct net_generic {
> + unsigned int len;
> + struct rcu_head rcu;
> +
> + void *ptr[0];
> +};
> +
> +static inline void *net_generic(struct net *net, int id)
> +{
> + struct net_generic *ng;
> + void *ptr;
> +
> + rcu_read_lock();

> + ng = rcu_dereference(net->gen);
> + BUG_ON(id == 0 || id > ng->len);
> + ptr = ng->ptr[id - 1];
> + rcu_read_unlock();
> +
> + return ptr;
> +}
> +
> +extern int net_assign_generic(struct net *net, int id, void *data);
> +#endif
> diff --git a/net/core/net_namespace.c b/net/core/net_namespace.c
> index 7ef3bac..b384840 100644
> --- a/net/core/net_namespace.c
> +++ b/net/core/net_namespace.c
> @@ -7,6 +7,7 @@
> #include <linux/sched.h>
> #include <linux/idr.h>
> #include <net/net_namespace.h>
> +#include <net/netns/generic.h>
>
> /*
>   * Our network namespace constructor/destructor lists
> @@ -21,6 +22,8 @@ LIST_HEAD(net_namespace_list);
> struct net init_net;
> EXPORT_SYMBOL(init_net);
>
> +#define INITIAL_NET_GEN_PTRS 13 /* +1 for len +2 for rcu_head */
> +
> /*
>   * setup_net runs the initializers for the network namespace object.
>   */
> @@ -29,10 +32,21 @@ static __net_init int setup_net(struct net *net)
>   /* Must be called with net_mutex held */
>   struct pernet_operations *ops;
>   int error;
> + struct net_generic *ng;
>
>   atomic_set(&net->count, 1);
>   atomic_set(&net->use_count, 0);
>
> + error = -ENOMEM;
> + ng = kzalloc(sizeof(struct net_generic) +
> +   INITIAL_NET_GEN_PTRS * sizeof(void *), GFP_KERNEL);

Why do you need to allocate more than sizeof(struct net_generic) ?

> + if (ng == NULL)
> + goto out;

> +
> + ng->len = INITIAL_NET_GEN_PTRS;
> + INIT_RCU_HEAD(&ng->rcu);
> + rcu_assign_pointer(net->gen, ng);
> +
> error = 0;
> list_for_each_entry(ops, &pernet_list, list) {
> if (ops->init) {
> @@ -54,6 +68,7 @@ out_undo:
> }
>
> rcu_barrier();
> + kfree(ng);
> goto out;
> }
>
> @@ -384,3 +399,50 @@ void unregister_pernet_gen_device(int id, struct pernet_operations
*ops)
> mutex_unlock(&net_mutex);
> }
> EXPORT_SYMBOL_GPL(unregister_pernet_gen_device);
> +
> +static void net_generic_release(struct rcu_head *rcu)
> +{
> + struct net_generic *ng;
> +
> + ng = container_of(rcu, struct net_generic, rcu);
> + kfree(ng);
> +}
> +
> +int net_assign_generic(struct net *net, int id, void *data)
> +{
> + struct net_generic *ng, *old_ng;
> +
> + BUG_ON(!mutex_is_locked(&net_mutex));
> + BUG_ON(id == 0);
> +
> + ng = old_ng = net->gen;

shouldn't it be rcu_dereferenced ?

> + if (old_ng->len >= id)
> + goto assign;
> +
> + ng = kzalloc(sizeof(struct net_generic) +
> + id * sizeof(void *), GFP_KERNEL);
> + if (ng == NULL)
> + return -ENOMEM;

```
> +
> + /*
> +  * Some synchronisation notes:
> +  *
> +  * The net_generic explores the net->gen array inside rcu
> +  * read section. Besides once set the net->gen->ptr[x]
> +  * pointer never changes (see rules in netns/generic.h).
> +  *
> +  * That said, we simply duplicate this array and schedule
> +  * the old copy for kfree after a grace period.
> +  */
> +
> + ng->len = id;
> + INIT_RCU_HEAD(&ng->rcu);
> + memcpy(&ng->ptr, &old_ng->ptr, old_ng->len);
> +
> + rcu_assign_pointer(net->gen, ng);
> + call_rcu(&old_ng->rcu, net_generic_release);
> +assign:
> + ng->ptr[id - 1] = data;
> + return 0;
> +}
> +EXPORT_SYMBOL_GPL(net_assign_generic);
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 2/14][NETNS]: Generic per-net pointers.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 14:09:57 GMT
View Forum Message <> Reply to Message

[snip]

```
>> @@ -29,10 +32,21 @@ static __net_init int setup_net(struct net *net)
>>   /* Must be called with net_mutex held */
>>   struct pernet_operations *ops;
>>   int error;
>> + struct net_generic *ng;
>>
>>   atomic_set(&net->count, 1);
>>   atomic_set(&net->use_count, 0);
>>
>> + error = -ENOMEM;
>> + ng = kzalloc(sizeof(struct net_generic) +
>> +   INITIAL_NET_GEN_PTRS * sizeof(void *), GFP_KERNEL);
```

>
> Why do you need to allocate more than sizeof(struct net_generic) ?

That's just an optimization to avoid many reallocations in the
nearest future. I planned to make similar in net_assign_generic
(allocate a bit more that required), but decided to do it later
to keep net_assign_generic simpler.

Currently I have only 5 users of generic pointers (tun and vlan
you see and I have patches for ipip, ipgre and sit tunnels), so
that's enough for the first time.

[snip]

>> +int net_assign_generic(struct net *net, int id, void *data)
>> +{
>> + struct net_generic *ng, *old_ng;
>> +
>> + BUG_ON(!mutex_is_locked(&net_mutex));
>> + BUG_ON(id == 0);
>> +
>> + ng = old_ng = net->gen;
>
> shouldn't it be rcu_dereferenced ?

Nope - nobody can race with us and change this pointer, so it's
safe to get one without rcu_dereference.

Thanks,
Pavel

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by paulmck on Fri, 11 Apr 2008 15:04:40 GMT
View Forum Message <> Reply to Message

On Fri, Apr 11, 2008 at 11:55:59AM +0400, Pavel Emelyanov wrote:
> Paul E. McKenney wrote:
> > On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:
> >> This is the first step in making tuntap devices work in net
> >> namespaces. The structure mentioned is pointed by generic
> >> net pointer with tun_net_id id, and tun driver fills one on
> >> its load. It will contain only the tun devices list.
> >>

> >> So declare this structure and introduce net init and exit hooks.
> >
> > OK, I have to ask... What prevents someone else from invoking
> > net_generic() concurrently with a call to tun_exit_net(), potentially
> > obtaining a pointer to the structure that tun_exit_net() is about
> > to kfree()?
>
> It's the same as if the tun_net was directly pointed by the struct
> net. Nobody can grant, that the pointer got by you from the struct
> net is not going to become free, unless you provide this security
> by yourself.

So tun_net acquires some lock before calling net_generic(), and that
same lock is held when calling tun_exit_net()? Or is there but a
single tun_net task, so that it will never call tun_net_exit()
at the same time that it calls net_generic() for the tun_net pointer?

> But if you call net_generic to get some pointer other than tun_net,
> then you're fine (due to RCU), providing you play the same rules with
> the pointer you're getting.

Agreed, RCU protects the net_generic structure, but not the structures
pointed to by that structure.

> Maybe I'm missing something in your question, can you provide some
> testcase, that you suspect may cause an OOPS?

Just trying to understand what prevents one task from calling
net_generic() to pick up the tun_net pointer at the same time some other
task calls tun_net_exit().

Thanx, Paul

> >> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> >>
> >> ---
> >> drivers/net/tun.c | 53 ++-
> >> 1 files changed, 52 insertions(+), 1 deletions(-)
> >>
> >> diff --git a/drivers/net/tun.c b/drivers/net/tun.c
> >> index 7b816a0..9bfba02 100644
> >> --- a/drivers/net/tun.c
> >> +++ b/drivers/net/tun.c
> >> @@ -63,6 +63,7 @@
> >> #include <linux/if_tun.h>
> >> #include <linux/crc32.h>
> >> #include <net/net_namespace.h>
> >> +#include <net/netns/generic.h>

```
> >>
> >>  #include <asm/system.h>
> >>  #include <asm/uaccess.h>
> >> @@ -73,6 +74,11 @@ static int debug;
> >>
> >>  /* Network device part of the driver */
> >>
> >> +static unsigned int tun_net_id;
> >> +struct tun_net {
> >> + struct list_head dev_list;
> >> +};
> >> +
> >>  static LIST_HEAD(tun_dev_list);
> >>  static const struct ethtool_ops tun_ethtool_ops;
> >>
> >> @@ -873,6 +879,37 @@ static const struct ethtool_ops tun_ethtool_ops = {
> >>   .set_rx_csum = tun_set_rx_csum
> >>  };
> >>
> >> +static int tun_init_net(struct net *net)
> >> +{
> >> + struct tun_net *tn;
> >> +
> >> + tn = kmalloc(sizeof(*tn), GFP_KERNEL);
> >> + if (tn == NULL)
> >> +  return -ENOMEM;
> >> +
> >> + INIT_LIST_HEAD(&tn->dev_list);
> >> +
> >> + if (net_assign_generic(net, tun_net_id, tn)) {
> >> +  kfree(tn);
> >> +  return -ENOMEM;
> >> + }
> >> +
> >> + return 0;
> >> +}
> >> +
> >> +static void tun_exit_net(struct net *net)
> >> +{
> >> + struct tun_net *tn;
> >> +
> >> + tn = net_generic(net, tun_net_id);
> >> + kfree(tn);
> >> +}
> >> +
> >> +static struct pernet_operations tun_net_ops = {
> >> + .init = tun_init_net,
> >> + .exit = tun_exit_net,
```

```
> >> +};
> >> +
> >>  static int __init tun_init(void)
> >>  {
> >>   int ret = 0;
> >> @@ -880,9 +917,22 @@ static int __init tun_init(void)
> >>   printk(KERN_INFO "tun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
> >>   printk(KERN_INFO "tun: %s\n", DRV_COPYRIGHT);
> >>
> >> + ret = register_pernet_gen_device(&tun_net_id, &tun_net_ops);
> >> + if (ret) {
> >> +  printk(KERN_ERR "tun: Can't register pernet ops\n");
> >> +  goto err_pernet;
> >> + }
> >> +
> >>   ret = misc_register(&tun_miscdev);
> >> - if (ret)
> >> + if (ret) {
> >>    printk(KERN_ERR "tun: Can't register misc device %d\n", TUN_MINOR);
> >> +  goto err_misc;
> >> + }
> >> + return 0;
> >> +
> >> +err_misc:
> >> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
> >> +err_pernet:
> >>   return ret;
> >>  }
> >>
> >> @@ -899,6 +949,7 @@ static void tun_cleanup(void)
> >>   }
> >>   rtnl_unlock();
> >>
> >> + unregister_pernet_gen_device(tun_net_id, &tun_net_ops);
> >>  }
> >>
> >>  module_init(tun_init);
> >> --
> >> 1.5.3.4
> >>
> >> --
> >> To unsubscribe from this list: send the line "unsubscribe netdev" in
> >> the body of a message to majordomo@vger.kernel.org
> >> More majordomo info at  http://vger.kernel.org/majordomo-info.html
> > --
> > To unsubscribe from this list: send the line "unsubscribe netdev" in
> > the body of a message to majordomo@vger.kernel.org
> > More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

> >
>

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 15:22:51 GMT
View Forum Message <> Reply to Message

Paul E. McKenney wrote:
> On Fri, Apr 11, 2008 at 11:55:59AM +0400, Pavel Emelyanov wrote:
>> Paul E. McKenney wrote:
>>> On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:
>>>> This is the first step in making tuntap devices work in net
>>>> namespaces. The structure mentioned is pointed by generic
>>>> net pointer with tun_net_id id, and tun driver fills one on
>>>> its load. It will contain only the tun devices list.
>>>>
>>>> So declare this structure and introduce net init and exit hooks.
>>> OK, I have to ask... What prevents someone else from invoking
>>> net_generic() concurrently with a call to tun_exit_net(), potentially
>>> obtaining a pointer to the structure that tun_exit_net() is about
>>> to kfree()?
>> It's the same as if the tun_net was directly pointed by the struct
>> net. Nobody can grant, that the pointer got by you from the struct
>> net is not going to become free, unless you provide this security
>> by yourself.
>
> So tun_net acquires some lock before calling net_generic(), and that
> same lock is held when calling tun_exit_net()? Or is there but a

No.

> single tun_net task, so that it will never call tun_net_exit()
> at the same time that it calls net_generic() for the tun_net pointer?

tun_net_exit is called only when a struct net is no longer referenced
and is going to be kfree-ed itself, so it's impossible (or BUGy by its
own) that someone still has a pointer on this net.

Providing the struct net is alive (!), the net->gen array is alive (or
is scheduled for kfree after RCU grace period). Thus, if your code
holds the net and uses the net_generic() call, then it will get alive
net->gen array and alive tun_net pointer.

Next, what happens after net_generic() completes and leaves the RCU-read
section? Simple - the struct net is (should be) still referenced, so the
tun_net_exit cannot yet be called and thus the tun_net pointer obtained
earlier is alive. Unlike the (possibly) former instance of the net_generic
array, but nobody references this one in my code (and should not do so,
hm... I think I'll add this rule to the comments).

>> But if you call net_generic to get some pointer other than tun_net,
>> then you're fine (due to RCU), providing you play the same rules with
>> the pointer you're getting.
>
> Agreed, RCU protects the net_generic structure, but not the structures
> pointed to by that structure.

They are protected by struct net reference counting.

>> Maybe I'm missing something in your question, can you provide some
>> testcase, that you suspect may cause an OOPS?
>
> Just trying to understand what prevents one task from calling
> net_generic() to pick up the tun_net pointer at the same time some other
> task calls tun_net_exit().

If this task dereferences a "held" struct net, then should be OK. If
this task does not, this will OOPs in any case.

Consider the struct net to look like

```
struct net {
 ...
 void *ptrs[N];
}
```

and the net_generic to be just

```
static inline void net_generic(struct net *net, int id)
{
 BUG_ON(id >= N);
 return net->ptrs[id - 1];
}
```

That's the same to what I propose, except for the ptrs array is on the
RCU protected memory.

> Thanx, Pau

Thanks,
Pavel

Subject: Re: [PATCH 0/14 (3 subsets)] Make tuns and vlans devices work per-net.
Posted by Daniel Lezcano on Fri, 11 Apr 2008 15:42:54 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Hi, guys.
>
> I've recently sent a TUN devices virtualization, but it was rejected
> by Dave, since the struct net is becoming a dumping ground.
>
> I agree with him - we really need some way to register on-net data
> dynamically. That's my view of such a thing and two examples of how
> to use it (TUN and VLAN devices virtualization).
>
> If this will be found good, I'll send these sets to David, hoping he
> will accept them :)

Pavel,

seems to be a smart solution :)

I am just afraid with the performances when the network resources are to be accessed in the fast path like a routing table (that seems not to be the case for tun and vlan). Shall we assume the fast path should always go to struct net and non critical path can go to net_generic ?

-- Daniel

Subject: Re: [PATCH 0/14 (3 subsets)] Make tuns and vlans devices work per-net.
Posted by Pavel Emelianov on Fri, 11 Apr 2008 15:57:25 GMT
View Forum Message <> Reply to Message

Daniel Lezcano wrote:
> Pavel Emelyanov wrote:
>> Hi, guys.
>>

>> I've recently sent a TUN devices virtualization, but it was rejected
>> by Dave, since the struct net is becoming a dumping ground.
>>
>> I agree with him - we really need some way to register on-net data
>> dynamically. That's my view of such a thing and two examples of how
>> to use it (TUN and VLAN devices virtualization).
>>
>> If this will be found good, I'll send these sets to David, hoping he
>> will accept them :)
>
> Pavel,
>
> seems to be a smart solution :)

Thanks :) However, I've already found a bug in the 1st patch (already fixed).

> I am just afraid with the performances when the network resources are to
> be accessed in the fast path like a routing table (that seems not to be
> the case for tun and vlan). Shall we assume the fast path should always
> go to struct net and non critical path can go to net_generic ?

Hm... I put call to net_generic() into tunnels rcv call and measured
the performance with netperf - no performance penalty. I tried to make
net_generic() work w/o any locks and looks like I've managed to make
it fast enough :)

I think, that core kernel code and protocols should/may use the struct
net, while modules are better to work via generic pointers. However, if
the generic pointers cause noticeable performance degradation, then we
may ask Dave to bear with on-net members :)

>   -- Daniel
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by paulmck on Fri, 11 Apr 2008 16:27:52 GMT
View Forum Message <> Reply to Message

On Fri, Apr 11, 2008 at 07:45:06PM +0400, Pavel Emelyanov wrote:
> Paul E. McKenney wrote:
> > On Fri, Apr 11, 2008 at 11:55:59AM +0400, Pavel Emelyanov wrote:
> >> Paul E. McKenney wrote:

> >>> On Thu, Apr 10, 2008 at 07:06:24PM +0400, Pavel Emelyanov wrote:
> >>>> This is the first step in making tuntap devices work in net
> >>>> namespaces. The structure mentioned is pointed by generic
> >>>> net pointer with tun_net_id id, and tun driver fills one on
> >>>> its load. It will contain only the tun devices list.
> >>>>
> >>>> So declare this structure and introduce net init and exit hooks.
> >>> OK, I have to ask... What prevents someone else from invoking
> >>> net_generic() concurrently with a call to tun_exit_net(), potentially
> >>> obtaining a pointer to the structure that tun_exit_net() is about
> >>> to kfree()?
> >> It's the same as if the tun_net was directly pointed by the struct
> >> net. Nobody can grant, that the pointer got by you from the struct
> >> net is not going to become free, unless you provide this security
> >> by yourself.
> >
> > So tun_net acquires some lock before calling net_generic(), and that
> > same lock is held when calling tun_exit_net()? Or is there but a
>
> No.
>
> > single tun_net task, so that it will never call tun_net_exit()
> > at the same time that it calls net_generic() for the tun_net pointer?
>
> tun_net_exit is called only when a struct net is no longer referenced
> and is going to be kfree-ed itself, so it's impossible (or BUGy by its
> own) that someone still has a pointer on this net.
>
> Providing the struct net is alive (!), the net->gen array is alive (or
> is scheduled for kfree after RCU grace period). Thus, if your code
> holds the net and uses the net_generic() call, then it will get alive
> net->gen array and alive tun_net pointer.
>
> Next, what happens after net_generic() completes and leaves the RCU-read
> section? Simple - the struct net is (should be) still referenced, so the
> tun_net_exit cannot yet be called and thus the tun_net pointer obtained
> earlier is alive. Unlike the (possibly) former instance of the net_generic
> array, but nobody references this one in my code (and should not do so,
> hm... I think I'll add this rule to the comments).
>
> >> But if you call net_generic to get some pointer other than tun_net,
> >> then you're fine (due to RCU), providing you play the same rules with
> >> the pointer you're getting.
> >
> > Agreed, RCU protects the net_generic structure, but not the structures
> > pointed to by that structure.
>
> They are protected by struct net reference counting.

Ah, OK, got it! Thank you for the tutorial!

> >> Maybe I'm missing something in your question, can you provide some
> >> testcase, that you suspect may cause an OOPS?
> >
> > Just trying to understand what prevents one task from calling
> > net_generic() to pick up the tun_net pointer at the same time some other
> > task calls tun_net_exit().
>
> If this task dereferences a "held" struct net, then should be OK. If
> this task does not, this will OOPs in any case.
>
> Consider the struct net to look like
>
> struct net {
> ...
> void *ptrs[N];
> }
>
> and the net_generic to be just
>
> static inline void net_generic(struct net *net, int id)
> {
> BUG_ON(id >= N);
> return net->ptrs[id - 1];
> }
>
> That's the same to what I propose, except for the ptrs array is on the
> RCU protected memory.

So RCU is protecting -only- the net_generic structure that net_generic()
is traversing, and the structure returned by net_generic() is protected
by a reference counter in the upper-level struct net.

If this is the approach, I am happy. ;-)

Thanx, Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 0/14 (3 subsets)] Make tuns and vlans devices work per-net.
Posted by davem on Fri, 11 Apr 2008 18:11:31 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Fri, 11 Apr 2008 19:57:25 +0400

> I think, that core kernel code and protocols should/may use the struct
> net, while modules are better to work via generic pointers. However, if
> the generic pointers cause noticeable performance degradation, then we
> may ask Dave to bear with on-net members :)

This sounds fine.

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by Pavel Emelianov on Sat, 12 Apr 2008 08:44:00 GMT
View Forum Message <> Reply to Message

> So RCU is protecting -only- the net_generic structure that net_generic()
> is traversing, and the structure returned by net_generic() is protected
> by a reference counter in the upper-level struct net.
>
> If this is the approach, I am happy. ;-)

Yup! Thank you :)

> Thanx, Paul
>

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by davem on Tue, 15 Apr 2008 07:44:17 GMT
View Forum Message <> Reply to Message

Pavel, please respin these patches since I applied the updated
version of patches 1 and 2 which added the generic netns bits.

Thanks!

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by Pavel Emelianov on Tue, 15 Apr 2008 10:00:33 GMT
View Forum Message <> Reply to Message

David Miller wrote:
> Pavel, please respin these patches since I applied the updated
> version of patches 1 and 2 which added the generic netns bits.

Thank you, David! I will do it today.

Are you willing to look at vlans and ip tunnels (ipip, gre, sit) as
well? This will be 6 sets ~7 patches each.

> Thanks!
>

Subject: Re: [PATCH 3/14][TUN]: Introduce the tun_net structure.
Posted by davem on Tue, 15 Apr 2008 10:06:34 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Tue, 15 Apr 2008 14:31:45 +0400

> Are you willing to look at vlans and ip tunnels (ipip, gre, sit) as
> well? This will be 6 sets ~7 patches each.

Sure.